Anssi Hiilinen

Datalaitteiden esikonfigurointijärjestelmä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

29.4.2016

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Anssi Hiilinen Datalaitteiden esikonfigurointijärjestelmä 29 sivua 29.4.2016
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Lehtori Ilpo Kuivanen Technical Specialist Kari Nevalainen

Tämän opinnäytetyön tarkoituksena on kehittää sovellus, jolla kootaan yhteen ja hallinnoidaan datalaitteiden konfiguroinnissa käytettäviä skriptejä. Sovelluksen lisäksi kehitetään mainittuja skriptejä käytettävyyden ja joustavuuden parantamiseksi. Datalaitteilla tarkoitetaan tässä yhteydessä reitittimiä ja kytkimiä.

Raportissa kuvataan kehitetyn sovelluksen käyttöliittymää, rakennetta ja toimintaa sekä kerrotaan, mitä esikonfiguroinnilla tarkoitetaan ja miten skriptit liittyvät siihen. Raportin pääpaino on kehitettävän sovelluksen rakenteen kuvauksessa.

Lopuksi kerrotaan kehitetyn järjestelmän käyttöönotosta ja arvioidaan, kuinka projektin toteutus onnistui.

Avainsanat	Esikonfigurointi, WinAPI, C++

Author(s) Title Number of Pages Date	Anssi Hiilinen CPE preconfiguration system 29 pages 29 April 2016
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Ilpo Kuivanen, Senior Lecturer Kari Nevalainen, Technical Specialist

The aim of this thesis was to develop a system that makes preconfiguration of customer premises equipment such as routers and switches easier, faster and more efficient. The system consists of two parts: Tera Term macros that are used in preconfiguring and an application to view and launch these macro scripts.

The main focus of the report is in describing the structure of the developed application called esaM. Preconfiguration as a process is also described as well as the macros used in preconfiguration.

Deployment of the developed system and analysis of the results of the project are described at the very end of the report.

Keywords	Preconfiguration, WinAPI, C++

Sisällys

Lyhenteet

TTL	Tera Term Language. Skriptikieli, jolla kirjoitetaan Tera Term MACROja.
COM-portti	RS-232-sarjaportti, Windowsissa nimellä COM1, COM2 jne. Sarjaportti, jonka avulla muodostetaan yhteys tietokoneen ja konfiguroitavan laitteen välille
DPI	Dots per inch. Mittayksikkö, pistettä tuumalla. Käytetään kuvaamaan kuvan tarkkuutta.
TFTP	Trivial file transfer protocol. Tiedostonsiirtomenetelmä.
DSL	Digital subscriber line. Suomennettuna: digitaalinen tilaajayhteys. Nimitys joukolle tietoliikennetekniikoita.
WinAPI	Windows application programming interface. Microsoft Windowsin ohjelmointirajapinta.

1 Johdanto

Tämän työn tarkoituksena on kehittää järjestelmä, jonka avulla Posti Oy:n asiakasyrityksen datalaitteiden esikonfigurointia voidaan helpottaa ja yksinkertaistaa. Esikonfigurointi suoritettiin aiemmin asiakasyrityksen toimesta, mutta toiminta sekä työntekijät siirrettiin yhteistyön tiivistämisen yhteydessä Postille. Samalla ilmeni tarve kehittää datalaitteiden konfigurointia varten uusi järjestelmä, sillä vanhan järjestelmän kehitystyö päättyi. Datalaitteilla tarkoitetaan tässä yhteydessä reitittimiä ja kytkimiä

Työhön kuuluu olemassa olevien Tera Term Makrojen kehittämistä sekä niiden käynnistämiseen ja ylläpitämiseen tarvittavan sovelluksen tuottaminen. Tavoitteena on kehittää yhtenäinen kokonaisuus, joka on siirreltävissä ja muokattavissa mahdollisimman pienellä vaivalla.

Kehitettävä sovellus on toiminnaltaan varsin yksinkertainen. Tarkoituksena on, että käyttäjä voi käyttöliittymän painiketta painamalla käynnistää makron, joka konfiguroi halutun laitteen. Tämän lisäksi sovelluksella voidaan vaikuttaa makrojen toimintaan mm. laitteisiin päivitettävää ohjelmistoversioita muuttamalla. Lisäksi on mahdollista tallettaa laitteiden ohjelmistopäivityksessä käytettäviä ip-osoitteita ja konsoliyhteyden avaamiseen käytettäviä konsoliportin numeroita, jolloin käyttäjän ei tarvitse syöttää niitä käsin aina, kun hän konfiguroi jonkin laitteen. Sovelluksesta on tarkoitus tehdä mahdollisimman joustava, jotta esimerkiksi uusia painikkeita voi lisätä koskematta lähdekoodiin lainkaan. Sovelluksesta kerrotaan luvussa 4.

Työn tavoitteena on kehittää esikonfigurointia niin, että siitä tulisi mahdollisimman helppoa ja vaivatonta, sekä niin, ettei siinä voisi tapahtua huolimattomuusvirheitä ja turhaa ajanhukkaa. Tavoitteiden täyttymisestä ja muista johtopäätöksistä kerrotaan raportin lopuksi luvussa 5.

Tässä raportissa esitetyt laitemallit, ohjelmistoversiot ja muut tiedot ovat päästä keksittyjä. Niiden tarkoitus on ainoastaan havainnollistaa kehitetyn sovelluksen ja Tera Term MACROjen toimintaa. Asiakasyrityksen pyynnöstä työssä ei mainita tarkkoja laitemalleja, ohjelmistoversioita eikä konfiguraatiotietoja.

2 Esikonfigurointi

Esikonfigurointi on prosessi, jossa asiakasyrityksen datalaitteeseen määritellään asetukset, joilla laitteen voi asentaa ja konfiguroida loppuun verkon yli. Tarkoituksena on siis se, että laitteen voi esikonfiguroinnnin jälkeen toimittaa asentajalle, joka käy vain kytkemässä laitteen asennuskohteeseen. Tämän jälkeen asiakasyritys voi itse konfiguroida laitteen käyttökuntoon verkon yli, eikä asentajan tarvitse suorittaa sen kummempia toimenpiteitä.

Konfiguroitavia laitteita on lukuisia erilaisia, ja kaikki laitteet toimivat hieman eri tavalla. Esikonfiguraatioon, eli laitteeseen määriteltäviin asetuksiin vaikuttaa laitemallin lisäksi käytettävä tekniikka. Samaa laitetta voidaan käyttää erilaisiin toteutustapoihin, kuten ADSL-, VDSL-, SHDSL-, tai ethernet-toteutuksiin, mutta esikonfiguraatio on tekniikasta riippuen kuitenkin eri. Tällaisiin laitteisiin, joilla voidaan toteuttaa eri tekniikoita, kytketään usein esikonfiguroinnin yhteydessä tarkoitukseen sopiva moduulikortti. Asiakasyrityksen pyynnöstä laitemalleista ja konfiguraatioista ei tässä raportissa kerrota tämän enempää.

Suurimmalle osalle konfiguroitavista laitteista tehdään kutakuinkin samat toimenpiteet.

- ohjelmistopäivitys
- mahdollisten lisenssien asennus
- konfigurointi.

Toimenpiteiden suorittaminen käsin on työn toisteisuuden vuoksi tarpeettoman työlästä ja aiheuttaa väistämättä enemmän tai vähemmän virheitä. Käsin konfiguroiminen on kuitenkin yleensä tarpeetonta, sillä useimmat toimenpiteet voidaan automatisoida niin, ettei käyttäjän tarvitse puuttua niiden suoritukseen mitenkään, ellei jotain mene pieleen. Esikonfigurointi voidaan nimittäin suorittaa käsin näppäilyn sijaan erinäisillä skripteillä, eli makroilla. Makrot tekevät työn tekemisestä helpompaa, nopeampaa ja ennen kaikkea vähentävät virheitä. Makrojen kirjoittamisessa on kuitenkin oltava huolellinen, sillä virheellinen makro aiheuttaa sen, että sama virhe toistetaan joka ikiseen konfiguroita-

vaan laitteeseen. Makroista ja niiden toiminnasta kerrotaan enemmän seuraavassa luvussa.

3 Tera Term MACRO

3.1 Tera Term

Tera Term on pääte-emulaattori Microsoft Windowsille. Sen avulla voidaan muodostaa yhteys konfiguroitavan laitteen sarjaporttiin. Tera Term tukee sitä varten kehitettyä skriptikieltä ”Tera Term Language”, lyhyemmin TTL. Tera Term Language skriptejä kutsutaan nimellä Tera Term MACRO. Tässä raportissa niihin viitataan sanalla makro. Makrojen avulla voidaan automatisoida erilaisia toimintoja. Datalaitteiden konfiguroinnissa makrot hoitavat laitteen kanssa kommunikoimisen niin, että käyttäjän ei tarvitse kirjoittaa käsin juuri mitään.

3.2 Tera Term Language

Tera Term Language on skriptikieli, jolla voidaan kirjoittaa Tera Term MACROja. TTL sisältää monia hyödyllisiä komentoja, joilla saadaan makrot toimimaan tehokkaammin ja varautumaan virhetilanteisiin. Hyödyllisimmät komennot ovat sendln ja wait, joiden avulla konfiguroitavalla laitteelle voidaan lähettää komentoja, ja odottaa niihin tiettyä vastausta. Wait-komennolla voi odottaa yhtä tai useampaa merkkijonoa. Result-muuttujan arvo määräytyy sen mukaan, mikä merkkijono saatiin. Esimerkiksi wait ”kissa” ”koira” ”karhu”, asettaisi resultin arvoksi numeron kaksi, jos merkkijono ”koira” luettaisiin. Jos timeout-arvo on määritelty joksikin nollaa suuremmaksi arvoksi, odotetaan vain niin kauan, kunnes on kulunut yhtä monta sekuntia. Jos odotettavaa merkkijonoa ei ole siihen mennessä saatu, tulee result-muuttujan arvoksi nolla ja makro jatkaa matkaansa[1]. Tera Term Language pitää sisällään myös komentoja, joilla voidaan lukea dataa tiedostosta, sekä merkkijonojen käsittelyyn käytettäviä komentoja. Kaikki komennot löytyvät TTL command referencestä[2].

3.3 Makrojen toiminta

Makrojen kehitystyö alkoi osaltani jo toissa kesänä, kun olin harjoittelijana nykyisellä asiakasyrityksellä esikonfiguroimassa datalaitteita. Makroja käytettiin jo silloin datalaitteiden konfiguroinnissa, mutta ne olivat vielä varsin yksinkertaisia. Mielestäni niitä olisi voinut kehittää pidemmälle ja sainkin varsin vapaat kädet muokata makroja mieleni mukaan. Makrojen kehittämisessä suurena apuna on TTL-komentojen dokumentaatio, TTL command reference[2]. Sinne on listattu kaikki TTL-kielen komennot ja selostettu selkeästi esimerkein, miten niitä käytetään.

Yksinkertaisimmillaan makrot eivät tee muuta kuin lähettävät komentoja laitteelle ja odottavat saavansa niihin vastauksen. Esimerkiksi laitteen käynnistyessä makro voi odottaa, kunnes se lukee puskurista rivin ”login:”, ja lähettää laitteelle ennalta määritellyn käyttäjänimen. Datalaitteiden konfigurointi onnistuisi näinkin yksinkertaisilla makroilla, mutta jotta niistä saataisiin enemmän hyötyä irti, on niiden oltava hieman tätä älykkäämpiä.

Tärkeä osa esikonfigurointia on laitteen ohjelmistopäivitys. Ohjelmistopäivityksiä varten konfiguroitavaan laitteeseen määritellään ip-osoite, jotta sen ohjelmistoversio voidaan päivittää tftp-palvelimelta. Oletetaan, että seitsemän ihmistä konfiguroi laitteita samaan aikaan. On hyvin todennäköistä, että jotkut näistä seitsemästä sattuvat päivittämään ohjelmistoa samaan aikaan ja syntyy ip-ristiriita. Tästä syystä jokaisella käyttäjällä on oltava oma ip-osoitteensa, jonka hän määrittää konfiguroitavaan laitteeseen. Tera Term MACROn inputbox-komennolla käyttäjältä voidaan pyytää syötteenä valinta, jolla käyttäjä tunnistetaan ja valitaan oikea ip-osoite. Tällöin käyttäjän olisi kuitenkin joka kerta syötettävä tämä tieto käsin kun hän konfiguroi laitetta. Tätä varten jokaisella käyttäjällä on paikallisessa hakemistossa tiedosto, johon on kirjattu ip-osoite. Yksi ja sama makro lukee ip-osoitteen esimerkiksi tiedostosta ”C:\tiedosto.txt”, ja koska kaikilla käyttäjillä on oma C:\-hakemistonsa, on ip-osoitekin eri. Näin kaikki käyttäjät voivat päivittää ohjelmistoja vaikka samaan aikaan, eikä ip-ristiriitoja synny, ellei joku ole asettanut itselleen epähuomiossa samaa osoitetta kuin jollain toisella.

Makrojen avulla voidaan ohjelmiston päivittämisen yhteydessä tarkistaa, onko laitteessa jo valmiiksi oikea ohjelmistoversio. Näin ei hukata aikaa ohjelmiston turhaan päivittämiseen, sillä ohjelmistopäivitys saattaa laitemallista riippuen kestää useita kymmeniä minuutteja. Tämän lisäksi on hyvä pitää huolta siitä, että päivityksen epäonnistuessa käyttäjä saa ilmoituksen eikä makro jatka matkaansa. Joissakin laitemalleissa on ollut

tapana ensin tyhjentää koko laite ja ladata vasta sitten uusi ohjelmistoversio. Jos tässä tapauksessa ohjelmistopäivitys menee pieleen ja makro jatkaa matkaansa käynnistääkseen laitteen uudelleen, ei laite enää käynnisty, sillä siitä puuttuu käynnistettävä ohjelmisto. Tällaisessa tapauksessa ohjelmisto on ladattava laitteeseen normaalista poikkeavalla tavalla. Se taas voi olla työläs ja aikaa vievä prosessi.

Makrot voivat myös muuttaa toimintaansa kysymällä laitteelta tietoja. Jotkin konfiguroitavat laitteet ovat modulaarisia laitteita, eli samaan runkoon voidaan käytettävästä tekniikasta riippuen kytkeä erilaisia moduulikortteja. Toteutustapa yleensä määrää myös käytettävän esikonfiguraation. Esimerkiksi ADSL-totetutustavalla esikonfiguraatio on erilainen kuin SHDSL-toteutustavalla, vaikka konfiguroitava laite olisi täysin samanmallinen. Makro voikin kysyä laitteelta, mitä moduulikortteja siihen on kytketty, ja päättää itsenäisesti asetettavan esikonfiguraation. Tällä tavoin voidaan edelleen vähentää käyttäjältä tarvittavia syötteitä ja sitä kautta myös virheitä.

Ensimmäinen askel laitteen konfiguroinnissa on kuitenkin konsoliyhteyden muodostaminen. Se onnistuu TTL-komennolla ”connect” seuraavalla tavalla: ”connect '/C=x'”, jossa x on halutun COM-portin numero[3]. Aiemmin COM-portin numero kysyttiin inputbox-ikkunalla hieman samaan tapaan kuin ip-osoitekin. Makroilla voidaan kuitenkin lukea tiedostoista tekstiä, joten käyttäjältä kysymisen sijaan tiedot voidaan kirjoittaa tiedostoon. Näin käyttäjän ei tarvitse joka kerta antaa makrolle käsin samaa tietoa, vaan se osaa itse lukea tarvitsemansa tiedot tiedostosta. Tällä säästetään sekä aikaa että vaivaa. Käyttäjä voi tehdä jotakin muuta hyödyllistä sen sijaan, että hän kyttää terminaali-ikkunan ääressä odottamassa, koska makro kysyy häneltä tietoja.

Kun makroihin lisätään käyttäjätietojen lukeminen, laitemallin tunnistus, virheiden tarkistus ja muut käytettävyyttä parantavat seikat, tulee niistä helposti vaikeasti luettavia. Vaikeaselkoisuus tuo mukanaan ongelmia, kun tulee tarve muokata makroa. Yksi tällainen tilanne on, kun jonkin laitemallin haluttu ohjelmistoversio muuttuu.

Päivitettävä ohjelmistoversio oli aiemmin tapana kirjoittaa suoraan makroon, josta sitä käytiin tarpeen tullen muuttamassa. Makroja on kuitenkin varsin paljon, ja ohjelmistoversion etsiminen makron syövereistä voi olla hankalaa, jos ei ole itse kyseistä makroa kirjoittanut. Tästä syystä myös ohjelmistoversiot siirrettiin erilliseen tekstitiedostoon, josta ne voidaan lukea sekä käsin että ohjelmallisesti. Käytössä ollut Tera Termin jo hieman vanhentunut versio asetti tiettyjä rajoitteita, sillä merkkijonon pilkkomiseen käytettävä strsplit-komento vaatii Tera Term -version 4.67 [4]. Tästä syystä tarvittavat tie-

dot on kirjattava kaikki omille riveilleen, jotta niitä voidaan käsitellä erillisinä tietoina. Ohjelmistoversiot onkin kirjattu tiedostoon niin, että ensimmäisellä rivillä on tunnistetieto, josta käy ilmi laitemalli. Sen jälkeen on kaksi riviä, joilla on versionumero sekä tftp-palvelimelta ladattavan ohjelmistotiedoston nimi. Makrot lukevat tätä tiedostoa rivi kerrallaan, kunnes löytävät rivin, joka vastaa makroon määriteltyä laitetunnusta. Tämän jälkeen makro lukee kahdelta seuraavalta riviltä talteen versionumeron, tiedoston nimen ja ryhtyy hommiin. Jos laitetunnusta ei löydy tiedostosta, annetaan käyttäjälle virheilmoitus messagebox-komennolla ja joko lopetetaan tai pyydetään käyttäjää syöttämään tarvittavat tiedot.

COM-portin numeron, ip-osoitteen ja ohjelmistoversioiden tallennus tiedostoon osoittautui hyödylliseksi ratkaisuksi, kun aloin työstää sovellusta, jolla makroja saadaan suoritettua. Ohjelmistoversioita voidaan hallinnoida yksinkertaisella graafisella käyttöliittymällä ja ip-osoitetta sekä COM-porttia pudotusvalikolla. Sovelluksessa onkin mahdollista luoda itselleen COM/IP-profiileja, joita vaihtamalla voi nopeasti muuttaa makrojen käyttämää COM-portin numeroa sekä ip-osoitetta. COM/IP-profiiliksi kutsutaan siis konsoliportin numeron ja ip-osoitteen yhdessä muodostamaa profiilia, jota käytetään makroissa. Tiedostot toimivat sovelluksen ja makrojen rajapintana, vaikka ne eivät olekaan muuten toisiinsa kytköksissä. Makroja voi suorittaa sellaisenaankin vallan mainiosti, mutta sovellus tarjoaa selkeän näkymän, johon ne voidaan niputtaa helposti saataville ja selkeään järjestykseen.

4 esaM

Kehitettävä sovellus on nimeltään esaM. Nimi on mielikuvitusta säästäen johdettu sanoista esiasennus ja makrot. Sovelluksen tarkoitus on tarjota käyttäjälle yhteen näkymään kokoelma painikkeita, joiden avulla käynnistetään laitteiden konfigurointiin käytettäviä makroja. Sovelluksen kehitystyö alkoi vuodenvaihteessa kokeilumielessä. Alun perin tarkoituksena oli vain kokeilla, olisiko mahdollista tehdä sovellus, jonka käyttöliittymää voisi muokata erillisellä konfiguraatiotiedostolla koskematta lähdekoodiin lainkaan. Kokeilu onnistui, ja päätin kehittää kunnollisen ja toimivan sovelluksen. Ohjelmoinnin alkuvaiheessa koodi syntyi yrityksen ja erehdyksen kautta, joten koko ohjelmakoodi on pariinkin otteeseen tullut kirjoitettua uusiksi, sillä alkuperäisen koodin muokkaus kävi lopulta mahdottomaksi sen ollessa liian sekavaa ja testaamatonta.

4.1 Ominaisuudet

Sovellus tuli saada käyttökuntoon varsin tiukalla aikataululla, joten ominaisuuksia ei ensi alkuun ollut montaa. Ensimmäisessä loppukäyttäjien testattavassa versiossa oli ainoastaan painikkeita makrojen käynnistämiseksi. Valikkoa, taustakuvaa tai muita ominaisuuksia ei ollut. Kun aiemmat ominaisuudet tulivat siihen kuntoon, ettei niitä enää tarvinnut viilata paremmaksi, kehitettiin sovellukseen lisää ominaisuuksia. Toistaiseksi sovelluksella on seuraavat ominaisuudet.

- painikkeiden lisäys ja poisto konfiguraatiotiedostolla
- päyttäjäkohtaisten painikkeiden lisäys ja poisto painikkein
- käytettävän konsoliportin ja ip-osoitteen valinta ja tallennus
- datalaitteisiin päivitettävien ohjelmistoversioiden muokkaus ja hallinta.

Kehitystyö ei kuitenkaan pääty tähän. Jos käyttäjillä tai itselläni ilmenee ideoita tai tarvetta uusille ominaisuuksille, otetaan ne kehitystyön alle ja lisätään sovelluksen tuleviin versioihin. Yksi tällainen mahdollinen ominaisuus on näkymän vaihtaminen. Tällä hetkellä sovellus tarjoaa yhden näkymän, jossa on konfiguraatiotiedostossa määritellyt painikkeet. Jos tulisi esimerkiksi tilanne, että asiakasyrityksiä tulisi lisää, voisi sovellukseen lisätä ominaisuuden, jolla tuota näkymää voi vaihtaa. Näin eri asiakkailla olisi omat näkymänsä ja konfiguraatiotiedostonsa. Käyttäjä voisi valita esimerkiksi pudotusvalikolla näkymän ”Asiakas A”, jolloin ikkunaan piirtyisi asiakkaan A datalaitteiden konfigurointiin soveltuvien makrojen painikkeet. Asiakas B:llä taas olisi erilainen näkymä ja eri makrot. Samalla periaatteella näkymää voidaan nytkin rajata käyttämällä eri konfiguraatiodietostoja. Kesätyöntekijän tai muun tuuraajan ei välttämättä tarvitse nähdä kaikkia painikkeita, vaan hänelle voidaan tarjota näkymä, jossa on esillä vain niiden laitteiden painikkeet, joita hän osaa konfiguroida.

4.2 Toteutus

esaM on toteutettu käyttäen C++-ohjelmointikieltä ja WinAPI-ohjelmointirajapintaa. Se on siis ainoastaan Windows-ympäristössä toimiva sovellus, ja vaatii Windows Vistan tai uudemman käyttöjärjestelmän toimiakseen. Kehitysympäristönä toimii Visual Studio

2013 ja käyttöjärjestelmänä 64-bittinen Windows 10 Professional. Sovellusta on tarkoitus käyttää 32- ja 64-bittisillä Windows 7 Enterprise -käyttöjärjestelmillä ja sen käyttöliittymän tulee sopia myös varsin pienelle kannettavan tietokoneen näytölle. Tiedonlähteenä ohjelmoinnissa toimii pääasiassa Microsoft Developer Network sekä joitakin tiedonmurusia kirjasta Programming Windows, Fifth Edition, jonka on kirjoittanut Charles Petzold.

Ohjelmointikielen valinnassa suurimpana tekijänä oli kiinnostus ja halu oppia kirjoittamaan graafisella käyttöliittymällä varustettuja Windows-sovelluksia alusta loppuun. Sovelluksen piti alun perin olla vain omaksi iloksi tehty kokeilu. Lisäksi ohjelman tuli olla helppo asentaa ja tutun oloinen kokemattomammallekin tietokoneen käyttäjälle. Tähän tarkoitukseen C++ ja Visual Studio sopivat mielestäni hyvin, joten en varsinaisesti edes harkinnut mitään muuta vaihtoehtoa.

4.2.1 WinAPI

Windows-ohjelmointi eroaa merkkipohjaisten sovellusten ohjelmoinnista varsin merkittävästi. Sovelluksessa painikkeet, tekstikentät, ikkunat ja oikeastaan kaikki ovat ikkunoita. Ikkunat lähettävät ja vastaanottavat viestejä, joiden perusteella ne tietävät miten niiden tulee missäkin tilanteessa toimia[5]. Jokaisella ikkunalla on ikkunaproseduuri, window procedure, jossa päätetään miten viestiin reagoidaan. Viestejä luetaan viestijonosta viestisilmukassa, eli message loopissa. Viestisilmukka koostuu yleensä kolmesta funktiosta.

- GetMessage hakee viestin jonosta ja kopioi sen MSG-tyyppiseen tietueeseen. Jos viesti on WM_QUIT, funktio palauttaa FALSE, eli 0. WM_QUIT on yleensä ensimmäinen askel ohjelman sulkemisessa.

- TranslateMessage tulkitsee näppäimenpainallukset ja lähettää WM_CHAR –viestin viestijonoon sikäli, kun painallus on ASCII-merkki.

- DispatchMessage kutsuu viestissä määritellyn ikkunan ikkunaproseduuria argumenteilla, jotka GetMessage luki MSG-tyyppiseen tietueeseen.

esaM-sovelluksen viestisilmukka näyttää seuraavalta:

```
while (GetMessage(&msg, NULL, 0, 0))
{
        if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
        {
                TranslateMessage(&msg);
                DispatchMessage(&msg);
        }
}
```

Tässä silmukassa on aiemmin mainittujen lisäksi myös TranslateAccelerator-funktio, joka vertaa viestiä accelerator-tauluun, ja tarkistaa, onko näppäinyhdistelmälle määritelty jokin erityinen tehtävä. Tässä tapauksessa esimerkiksi alt+shift+7 avaa ohjelman about-dialogin. TranslateMessage ja DispatchMessage suoritetaan siis vain, jos viestiä ei löydy accelerator-taulukosta, eli TranslateAccelerator palauttaa arvon 0[6]. Windows-ohjelmoinnissa FALSE on yhtä kuin luku 0, ja TRUE taas mikä tahansa muu luku [7].

Ikkunaproseduurien tehtävänä on reagoida vastaanottamiinsa viesteihin. Ikkunat vastaanottavat heti luomisensa jälkeen WM_CREATE-viestin. Tämä viesti tulee vain kerran, ja esaM-sovelluksessa siihen reagoidaan pääikkunassa luomalla tarvittavat oliot ja alustamalla tiedot. Painikkeet aiheuttavat WM_COMMAND-viestin, johon reagoidaan ikkunaproseduurissa riippuen siitä, mikä painike on kyseessä. WM_COMMAND-viestin mukana tulee wParam-parametri, jolla viesti tunnistetaan. Sovelluksen jokaisella painikkeella ja valikon vaihtoehdolla on oma tunnuksensa, joka kulkee ikkunaproseduureille wParam-parametrina [8, 9].

EsaM käsittelee myös WM_TIMER-viestejä, joita ohjelmakoodissa määritelty ajastin lähettää 15 minuutin välein. Ajastimen tarkoituksena on päivittää näkymää automaattisesti, sillä sovellusta ei välttämättä suljeta moneen päivään. Jos esimerkiksi otetaan käyttöön uusi makro, pitää automaattinen näkymän päivitys huolen siitä, että uuden makron painike näkyy kaikille käyttäjille. Ajastimen luominen onnistuu SetTimer-funktiolla [10].

```
SetTimer(hWnd, IDT_TIMER, 900000, (TIMERPROC)NULL);
```

Ajastimelle voisi ohjelmoida oman proseduurinsa, mutta tässä tapauksessa ajastimen viestit käsitellään pääikkunan ikkunaproseduurissa. Ajastin on tuhottava käsin. Tämä tehdään WM_DESTROY-viestin käsittelyssä. WM-DESTROY on viesti, joka lähetetään ikkunalle kun se suljetaan. Pääikkunassa kutsutaan PostQuitMessage()-funktiota kun vastaanotetaan WM_DESTROY. Ennen tätä on kuitenkin hoidettava kaikki muu poistettava kuntoon. PostQuitMessage() aiheuttaa WM_QUIT-viestin lähetyksen, joka taas katkaisee viestisilmukan ja lopettaa ohjelman suorituksen.

Kaikkiin viesteihin ei välttämättä reagoida, mutta kaikki viestit on kuitenkin käsiteltävä. Tästä syystä ikkunaproseduurin on kutsuttava DefWindowProc-funktiota ja palautettava sen paluuarvo, jos kyseiselle viestille ei ole ohjelmoitu mitään muuta käsittelytapaa. EsaM ei käsittele lainkaan WM_PAINT-viestejä, mutta niitä ei voi kuitenkaan jättää huomiotta. Jos DefWindowProc-funktiota ei kutsuta, eivät myöskään painikkeet piirry ikkunaan oikein.

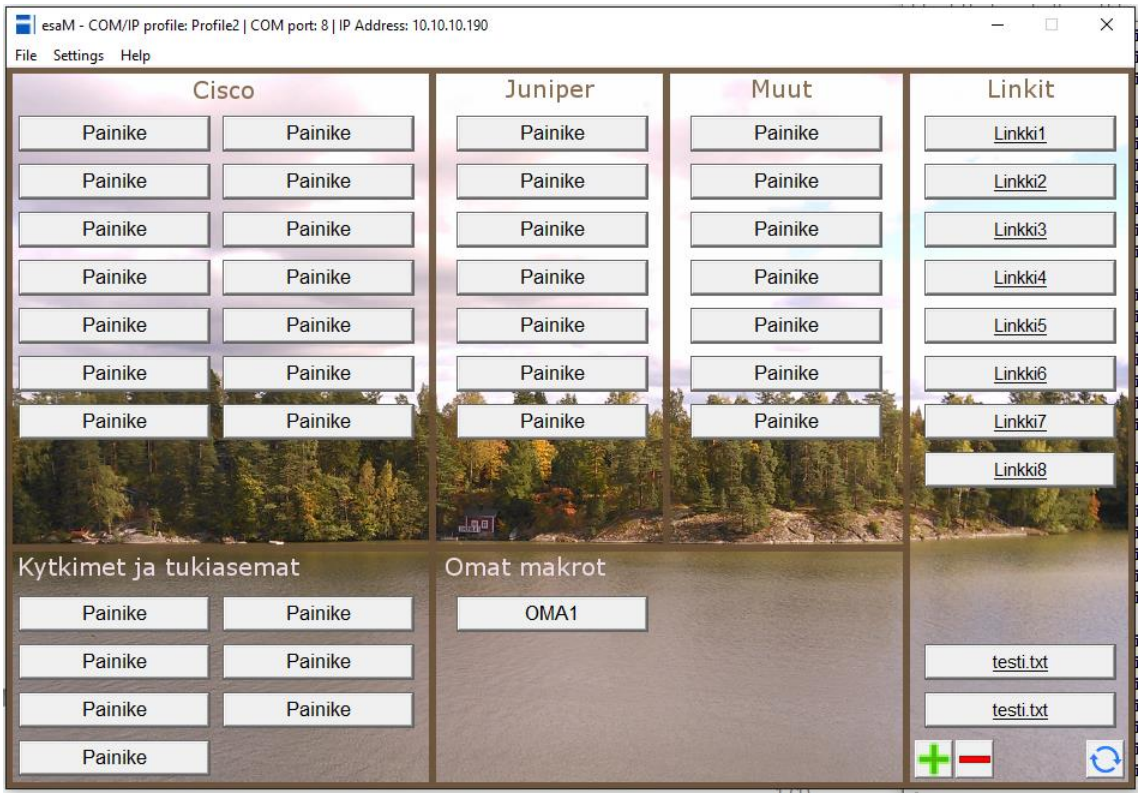
Dialogi-ikkunoiden ikkunaproseduuri on toiminnaltaan muuten samanlainen kuin muidenkin ikkunoiden, mutta WM_CREATE:n sijaan ne saavat WM_INITDIALOG-viestin kun ne on luotu. Dialogi-ikkunoiden ei myöskään täydy käsitellä kaikkia saamiaan viestejä, eli DefWindowProc-funktion kutsumista ei tarvita[6][11].

EsaM-sovelluksessa ikkunaproseduureja on kuusi kappaletta. WndProc-niminen proseduuri käsittelee pääikkunan viestit. Niitä ovat painikkeiden painallukset, valikon valinnat ja kaikki muut pääikkunalle tapahtuvat asiat. Tämän lisäksi sovelluksessa on omat ikkunaproseduurinsa jokaiselle ns. dialogi-ikkunalle. Dialogi-ikkunalla tarkoitetaan ikkunaa, jonka avulla käyttäjältä pyydetään syötettä tai näytetään tietoa. EsaM-sovelluksen dialogi-ikkunoista ja käyttöliittymästä kerrotaan tarkemmin kohdassa 4.2.5 Dialogi-ikkunat.

4.2.2 Käyttöliittymä

EsaM-sovelluksen käyttöliittymä on yksinkertainen ja selkeä. Siihen kootaan kaikki käytettävissä olevat makrot yhteen näkymään. Käyttöliittymä on pyritty suunnittelemaan niin, että käyttäjä pärjää normaalikäytössä yhdellä ikkunalla, eli

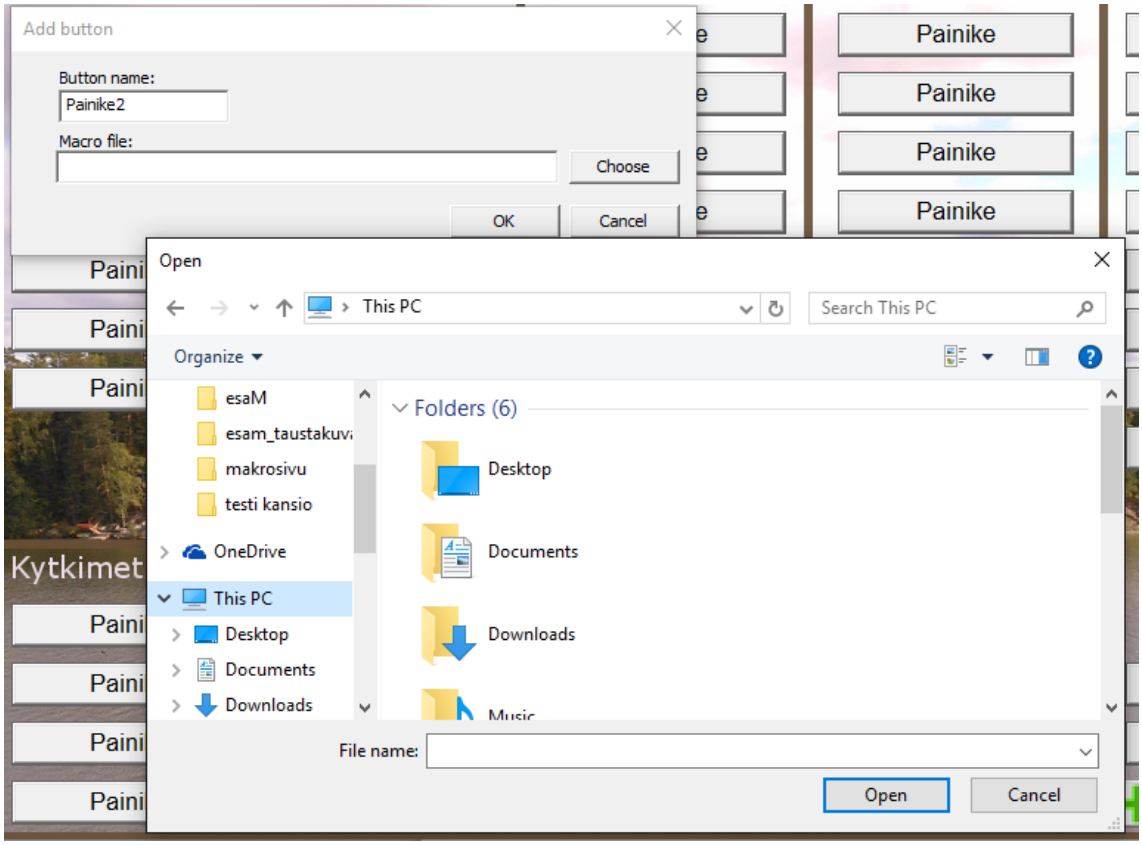
käytetyimpiä toiminnallisuuksia ei tarvitse etsiä useamman ikkunan takaa. Näkymästä on pyritty tekemään selkeä ja intuitiivinen.



Kuva 1. Sovelluksen käyttöliittymä

Kuvassa 1 näkyy sovelluksen käyttöliittymä. Suurin osa käytöstä tapahtuu näkymään piirrettyjen painikkeiden avulla. Jokainen painike suorittaa tietyn Tera Term MACRON. Painikkeet on jaettu ryhmiin sen perusteella, minkälaista laitetta niillä on tarkoitus konfiguroida. Oikea laita on varattu kokonaan painikkeille, joilla avataan linkkejä esimerkiksi tärkeille verkkosivuille tai tiedostoihin. Lisäksi tilaa on varattu käyttäjän omille makroille, joita voidaan lisätä oikean alanurkan +-nappulalla. Oikeasta alanurkasta löytyy myös painike omien painikkeiden poistamiseen, sekä painike, jolla näkymän saa päivitettyä. Ikkunan kokoa ei voi muuttaa, sillä sarakkeiden väliset viivat ja otsikkotekstit on piirretty taustakuvaan, ja ikkunan koon muuttaminen sotkisi näkymän. Myös kategorioiden nimet on piirretty suoraan taustakuvaan, joten niitä ei voi muuttaa ellei vaihda taustakuvaa. Taustakuva taas ei ole käyttäjän päätettävissä, vaan se on sisällytettynä sovelluksen ohjelmatiedostoon. Sovellukseen on ohjelmoitu mahdollisuus vaihtaa taustakuvaa, mutta se ei ole tällä hetkellä käytössä, sillä siitä ei toistaiseksi olisi mitään hyötyä.

Omien makrojen lisäystä varten on oma ikkunansa, jolla käyttäjä voi lisätä näkymään haluamansa makron. Ikkunaan käyttäjä syöttää tarvittavat tiedot, eli painikkeessa lukevan tekstin sekä Tera Term MACROn, jonka kyseinen painike suorittaa.



Kuva 2. Painikkeen lisäys

Kuvassa 2 esitetään painikkeen lisäys. Windowsin tiedostonavausikkuna aukeaa choose-painiketta painamalla, ja käyttäjä voi valita haluamansa tiedoston. Painamalla OK käyttäjä voi lisätä valitsemansa tiedoston näkymään sikäli, kun myös painikkeen nimi on syötetty. Molemmat kentät ovat pakollisia. Jos toinen tai kumpikin kenttä on tyhjä ja käyttäjä painaa OK, sulkeutuu ikkuna eikä näkymään lisätä mitään. Cancel-painikkeella mitään ei lisätä ja ikkuna suljetaan. Huomioitavaa kuvassa on myös se, että siinä näkyvä ”testi kansio” ei ole kirjoittajan osaamattomuutta, vaan tapa testata välilyöntejä tiedostopoluissa. Painikkeen poistaminen onnistuu lisäyspainikkeen viereisellä painikkeella. Kun käyttäjä klikkaa painiketta, menee sovellus poistotilaan ja yrittää poistaa seuraavaksi klikattavan painikkeen näkymästä. Jos käyttäjä painaa jotakin ”Omat makrot” -osiossa olevaa painiketta, se poistetaan ja siirrytään pois poistotilasta. Jos käyttäjä yrittää poistaa muita painikkeita, annetaan virheilmoitus ja poistutaan poistotilasta.

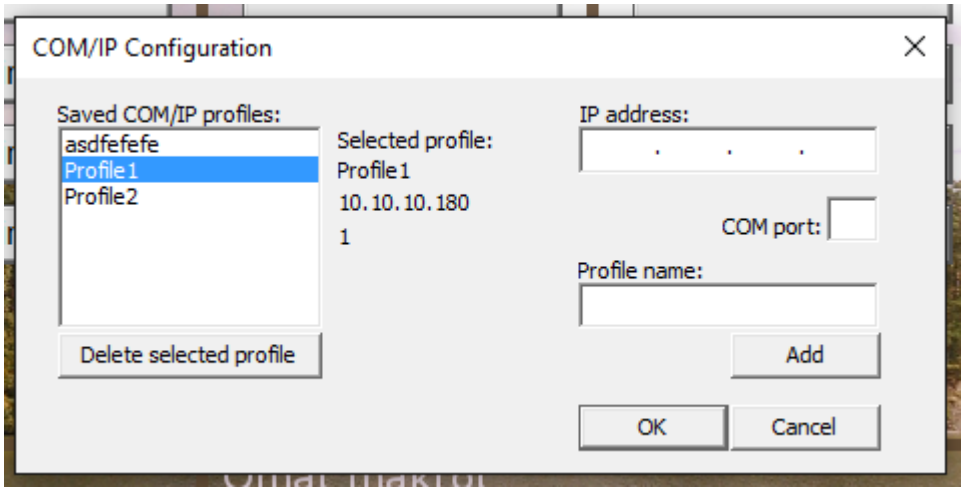
Poistotilan tunnistaa siitä, että miinusmerkkiä esittävä painike vaihtuu kuvan 3 kaltaiseksi.

Kuva 3. Poistotila

Perusnäkymän painikkeiden lisäksi käyttäjälle on tarjolla valikko, josta löytyy muutamia asetuksia. Käyttäjä voi luoda itselleen yhden tai useamman COM/IP-profiilin. COM/IP-profiili koostuu nimensä mukaisesti COM-portin numerosta ja ip-osoitteesta, joita käytetään makroissa konsoliyhteyden avaamiseen ja konfiguroitavan laitteen ohjelmistopäivitykseen. Tallennettujen COM/IP-profiilien tiedot kirjoitetaan tiedostoon u_info.txt. Tarvittavat tiedot ovat

- nimi
- konsoliportin numero
- ip-osoite
- tieto siitä, onko profiili valittuna.

Tiedot erotellaan toisistaan puolipistein, ja jokainen profiili on omalla rivillään. Tiedoston alkuun kirjoitetaan valitun profiilin konsoliportin numero ja ip-osoite molemmat omille riveilleen, sillä aiemmin esitellyt Tera Term MACROt lukevat ne siinä muodossa tiedoston alusta. U_info.txt mahdollistaa sen, että käyttäjä voi konfiguroida useampaa laitetta samanaikaisesti, sillä samalla COM-portilla ei voi kahta samanaikaista yhteyttä eikä samaa ip-osoitetta käyttää samassa verkossa. Kuvassa neljä näkyy dialogi-ikkuna, jolla käytettäviä COM/IP-profiileja voidaan lisätä, poistaa ja tarkastella. Kuvan vasemmassa reunassa on listaus tallennetuista profiileista. Valitun profiilin tiedot näytetään listan oikealla puolella. Profiilin voi poistaa painikkeella, joka on listan alapuolella, ja kuvan oikeassa laidassa näkyviin kenttiin voi kirjoittaa uuden profiilin tiedot ja lisätä sen listalle painamalla ”Add”-painiketta. Dialogilla ei voi muuttaa jo tallennettua profiilia, vaan se on ensin poistettava ja kirjoitettava sitten uusi. IP-osoitteen arvon tarkistamisesta vastaa käyttöjärjestelmä itse, sillä se kysytään WinAPI:n Ip Address Control -kentällä, joka ei ota vastaan kuin kelvollisia ip-osoitteita [12].



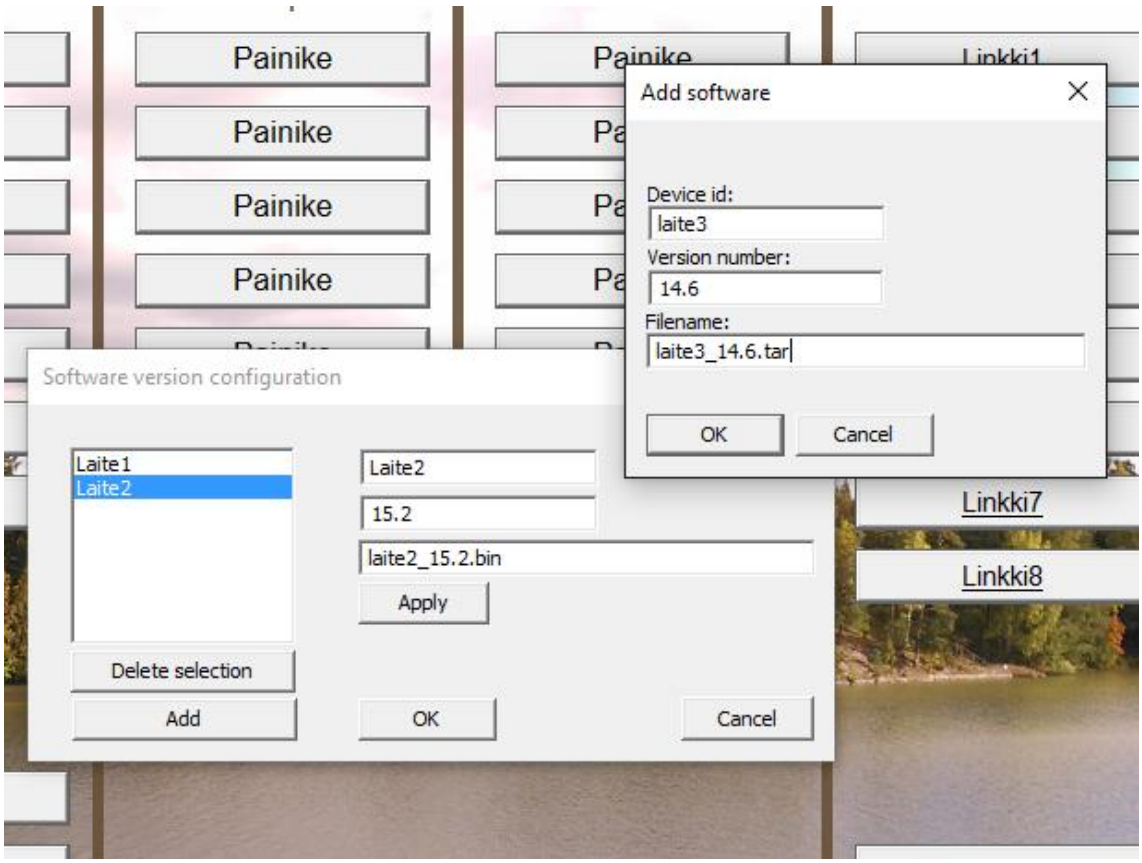
Kuva 4. COM/IP-asetusten hallinta.

Valitun profiilin tiedot näytetään sovelluksen otsikkopalkissa, jotta käyttäjä tietää, mitä tietoja käytetään, kun suoritetaan seuraava makro. Kuvassa 1 näkyy sovelluksen otsikkopalkki ja valitun profiilin tiedot. Profiilin vaihto tapahtuu Settings-pudotusvalikolla. Tallennetut profiilit näytetään allekkain, ja käyttäjä voi klikkaamalla valita mieleisensä profiilin. Profiilien tallentamiseen ja poistamiseen on oma ikkunansa, jonka avulla käyttäjä voi tarkastella tallennettuja profiileja, poistaa niitä sekä lisätä uusia.

Ohjelmistoversioiden muokkaus tapahtuu hieman samaan tapaan kuin COM/IP-profiilien muokkaus. Jokaiseen ohjelmistoversioon vaaditaan kolme tietoa

- Laitteen tunnus, Tera Term MACRO tunnistaa tämän perusteella oikean ohjelmiston.
- Ohjelmiston versionumero, makroissa tätä käytetään, kun tarkistetaan, onko laitteessa jo valmiiksi oikea ohjelmistoversio.
- Ladattavan tiedoston nimi. Tiedosto, joka ladataan konfiguroitavalle laitteelle tftp-palvelimelta ohjelmistopäivityksen yhteydessä.

Ohjelmistoversiot sisältävä tiedosto lukitaan esaM-sovelluksella niin, että kaksi käyttäjää ei epähuomiossa yritä muokata sitä samanaikaisesti. Lukon toiminnasta kerrotaan lisää kohdassa ”Sovelluksen rakenne”.



Kuva 5. Ohjelmistoversioiden hallinta

Kuvassa 5 esitetään dialogi-ikkunat, joilla ohjelmistoversioita hallitaan. Kuvassa päällimmäisenä olevalla dialogilla lisätään uusia ohjelmistoversioita. Alemmassa ikkunassa näkyy lista olemassa olevista laitteista. Listalta valitun laitteen tiedot näkyvät oikeanpuoleisissa tekstikentissä. Kentät ovat muokattavissa, ja ”Apply”-painiketta painamalla muutokset tallennetaan. Ohjelmistoversioiden hallinnassa tietoja siis voi muokata toisin kuin COM/IP-asetuksissa. Tämä johtuu siitä, että ohjelmistoversioita harvemmin joudutaan lisäämään uusia. Yleensä tilanne on se, että täytyy ainoastaan vaihtaa ladattava tiedosto sekä ohjelmiston versionumero. On tärkeää muistaa vaihtaa molemmat, sillä makrot käyttävät versionumeroa tarkistaakseen, onko konfiguroitavassa laitteessa jo valmiina oikea ohjelmistoversio. Jos käyttäjä haluaa poistaa listalta ohjelmistoversion, sovellus kysyy tähän vielä varmistusta yes/no-ikkunan avulla. Tämä siksi, että ohjelmistoversion vahingossa poistaminen johtaisi siihen, että kyseisen laitteen konfigurointiin käytettävä makro ei enää onnistuisi päivittämään siihen ohjelmistoa, tai tekemään mitään muutakaan. Makrot nimittäin antavat virheilmoituksen, jos eivät löydä itselleen määritettyä laitetunnusta tiedostosta, johon ohjelmistoversiot on kirjattu.

4.2.3 Sovelluksen rakenne ja luokat

Sovelluksessa on yksi pääluokka ja luokat painikkeiden, hallittavien ohjelmistoversioiden sekä käyttäjäkohtaisten asetusten muokkaukseen ja säilyttämiseen. Viestisilmukka ja ikkunaproseduurit suoritetaan pääluokassa, jossa myös luodaan muiden luokkien ilmentymät ennen pääikkunan piirtämistä. Nämä muiden luokkien ilmentymät, oliot, lukevat tietoa konfiguraatiotiedostoista ja tallentavat ne tietueisiin. Tietueita ei muokata suoraan pääluokassa, vaan niihin päästään käsiksi olioiden avulla. Jokaisessa luokassa on funktiot kaikkiin tarvittaviin toimenpiteisiin, kuten tiedon lukemiseen, muokkaamiseen, lisäämiseen, poistamiseen ja tallentamiseen. Tietueet ovat siis pääluokalta piilossa.

Btnconfig-niminen luokka vastaa painikkeiden tiedoista. Sovellus tarvitsee toimiakseen neljä konfiguraatiotiedostoa. Yksi niistä pitää sisällään tiedon muiden kolmen sijainnista ja sen tulee olla asennushakemistossa nimellä config.txt. Btnconfig-luokassa luetaan siitä muiden konfiguraatiotiedostojen tiedostopolut, sekä ttpmacro.exe:n tiedostopolku. Tämän jälkeen luetaan toisesta konfiguraatiotiedostosta kaikkien painikkeiden tiedot talteen. Painikkeiden tietoja voidaan lukemisen jälkeen hakea get-funktioilla. Käyttäjälle annetaan virheilmoitus, jos konfiguraatiotiedostoista puuttuu tietoja tai tiedot on kirjoitettu virheelliseen muotoon. Jos painikkeelle on annettu esimerkiksi liikaa tai virheellisiä parametreja, annetaan käyttäjälle virheilmoitus, jossa kerrotaan, millä rivillä virhe on. Linkit ja makrot säilötään samanlaisiin tietueisiin, jotka sijoitetaan kahteen vector-taulukkoon, linkit toiseen ja makrot toiseen. Painikkeilla on kaikilla seuraavat tiedot:

- wstring Name
- wstring Action
- wstring WorkingDir
- wstring X
- wstring Y
- bool userbutton.

Kaikki tiedot ”userbutton”-muuttujaa lukuun ottamatta ovat wstring-tyyppisiä. Wstring on merkkijonomuuttuja, joka perustuu char-tyyppisten alkioiden sijaan wchar_t-merkkeihin[14]. Wchar_t-tyyppisten merkkien kanssa tulee olla varovainen, sillä ne ovat eri järjestelmissä erikokoisia. Sizeof(wchar_t) ei siis ole kaikissa ympäristöissä sama. Tiedot luetaan merkkijonoihin siksi, että ne voidaan lukea tiedostosta sellaisenaan muuttujiin. X- ja Y-koordinaatit muutetaan kokonaisluvuiksi vasta siinä vaiheessa, kun niitä käytetään. Userbutton-muuttujalla kerrotaan, onko kyseessä käyttäjän määrittelemä painike. Omat painikkeet piirretään järjestyksessä kahteen sarakkeeseen, jotka näkyvät kuvan 1 kohdassa ”Omat makrot”. Käyttäjän omat makrot tallennetaan erilliseen tiedostoon, josta ne luetaan sen jälkeen, kun varsinaiset yhteiskäyttöiset makrot on luettu muistiin. Ne kuitenkin säilötään samaan taulukkoon kaikkien muidenkin painikkeiden kanssa, joten ne on merkittävä käyttäjän määrittelemiksi makroiksi, jotta ne

voidaan erottaa muista. Tämä on tärkeää esimerkiksi siinä vaiheessa, kun käyttäjä haluaa poistaa tai lisätä oman makron, poistamistoiminto ei salli poistaa painiketta, jonka ”isuserbutton”-arvo ei ole tosi. Name-muuttujaan säilötään se merkkijono, joka halutaan piirtää painikkeeseen.

Linkit piirretään kaikki samaan sarakkeeseen, ja niille voi määritellä ainoastaan Y-koordinaatin. Kuvan 1 oikeassa laidassa näkyy sarake, johon kaikki linkit piirretään. Konfiguraatiotiedostoon kirjoitettava numero kertookin, mille riville painike halutaan piirtää.

Btnconfig-luokka ei puutu painikkeiden ruudulle piirtämiseen millään tavalla. Sen tehtävänä on ainoastaan lukea painikkeiden tiedot ja pitää niitä yllä ohjelman suorituksen aikana. Painikkeiden tietojen kysymistä varten luokalla on lukuisia get-funktioita.

- std::wstring getTtpPath();
- **std::wstring getName(int idx, bool link);**
- **std::wstring getAction(int idx, bool link);**
- **std::wstring getWorkingDir(int idx, bool link);**
- std::wstring getSwFilePath();
- **int getX(int idx, bool link);**
- **int getY(int idx, bool link);**
- int getBtnCount();
- int getLinkCount();
- int getUsrBtnCount();

Listauksessa on lihavoituina ne funktiot, joilla haetaan yksittäisen painikkeen tietoja. Näissä käytetään indeksiä, jolla saadaan taulukosta oikea tieto, sekä totuusarvomuut-

tujaa link, jolla määritellään, mistä taulukosta tietoja on tarkoitus lukea. Jos link on tosi, on kyseessä linkki, ja jos se on epätosi, on kyseessä normaali painike. Lopuilla funktioilla haetaan tietoa, joka ei ole painikekohtaista, mutta jota tarvitaan painikkeiden piirtämiseen ja hallintaan. getTtpPath palauttaa nimensä mukaisesti tiedostopolun ttpmacro.exe-sovellukseen, getSwFilePath() palauttaa polun tiedostoon, josta löytyvät hallittavat ohjelmistoversiot. Kolme viimeistä palauttavat lukumäärät painikkeille, linkeille, sekä jo aiemmin mainituille käyttäjän määrittämille painikkeille. Painikkeiden määrää käytetään niiden piirtämisessä, kun halutaan käydä kaikki painikkeet läpi for-silmukassa. Tällöin kysytään btnconfig-luokalta painikkeiden kokonaismäärä ja käytetään sitä toistoehtona.

Get-funktioiden lisäksi btnconfig-luokassa on myös julkiset funktiot käyttäjäkohtaisten painikkeiden tiedostoon kirjoittamiseksi, lisäämiseksi sekä sieltä poistamiseksi. Näkymän päivittämistä varten on myös funktio, joka tyhjentää taulukot ja lukee konfiguraatiotiedostoista uudet tiedot. Yksityisiä funktioita luokassa ovat konfiguraatiotiedostojen lukemiseen käytettävät funktiot, sekä funktio, jolla pilkotaan makrotiedoston tiedostopolusta hakemiston polku irralleen.

swVersionConfig-luokassa käsitellään makrojen käyttämien ohjelmistoversioiden tietoja. Luokan konstruktorille annetaan parametrina btnconfig-luokan lukema tiedostopolku, jotta voidaan lukea ohjelmistoversiot muistiin. Ohjelmistoversiot sisältävässä tiedostossa on ensimmäisellä rivillä alkeellinen lukkomekanismi. Lukitsemattoman tiedoston ensimmäisellä rivillä on teksti ”OPEN”, ja lukitun tiedoston ensimmäisellä rivillä sen käyttäjän nimi, joka tietoja parhaillaan muokkaa. Käyttäjänimi saadaan Windowsin ympäristömuuttujasta USERNAME getenv-funktiolla [13]:

```
_wgetenv_s(&requiredSize, NULL, 0, L"USERNAME");
```

Jos tiedosto on lukittu, ilmoitetaan lukinneen käyttäjän nimi MessageBox-ikkunalla. Lukemisen onnistuessa kirjoitetaan ensimmäiselle riville sovelluksen käyttäjän käyttäjänimi. Luetut tiedot talletetaan kahteen eri taulukkoon. Toisessa pidetään alkuperäiset luetut tiedot ja toiseen tehdään muutoksia. Jos käyttäjä peruu tekemänsä muutokset, kirjoitetaan tiedostoon muokkaamattomat tiedot. Kun tietojen muokkaus lopetetaan, kirjoitetaan ensimmäiselle riville jälleen ”OPEN” ja sen jälkeen muokatut tiedot. Tällä pidetään huolta siitä, ettei kaksi eri käyttäjää muokkaa ohjelmistoversioita samaan aikaan.

Luetut ohjelmistoversiot talletetaan kahteen taulukkoon tietueina, jotka sisältävät laitetunnuksen, ohjelmiston versionumeron sekä ohjelmiston tiedostonimen. Taulukoita on kaksi, jotta ohjelmistoversioita käsittelevässä dialogi-ikkunassa voidaan perua kaikki jo tehdyt muutokset. Ohjelmistoversioiden muokkaaminen muuttaa tietueita taulukossa v_swversions, kun vaas v_swversions_backup säilyy muuttumattomana. Jos käyttäjä päättää asettaa muutokset voimaan, kirjoitetaan v_swversions takaisin tiedostoon, ja jos sen sijaan ei hyväksytä muutoksia, kirjoitetaan alkuperäinen tiedostosta luettu tieto takaisin tiedostoon muokkaamattomana. Tätä tarkoitusta varten on luokan funktio writeConfig. Funktio ottaa parametrinaan kaksi totuusarvoa. Ensimmäinen niistä kertoo aiotaanko tiedosto lukita vai avata. Tiedoston lukemisen yhteydessä tarkistetaan onko tiedosto lukossa, ja jos se ei ole, kutsutaan heti lukemisen jälkeen tätä writeConfig-funktiota sellaisin parametrein, että tiedostoon kirjoitetaan täsmälleen samat tiedot kuin sieltä luettiin, mutta lukitaan tiedosto. Toinen parametri taas päättää, kirjoitetaanko tiedostoon sieltä luettu tieto vaiko ohjelmassa muokattu tieto.

swVersionConfig-luokalla on seuraavat funktiot tietojen hakemiseen ja muokkaamiseen:

- std::wstring getName(int idx);
- std::wstring getVersion(int idx);
- std::wstring getFile(int idx);
- std::wstring getLockOwner();
- int getSize();
- bool setName(int idx, std::wstring newname);
- bool setVersion(int idx, std::wstring newversion);
- bool setFile(int idx, std::wstring newfile);
- bool addCpe(std::wstring name, std::wstring version, std::wstring file);

- bool delCpe(int idx);

Kuten aiemmin esitellyssä btnconfig-luokassa, myös tässä taulukoiden tietueita käsitellään indeksin perusteella. Indeksiä tarvitaan aina, kun käsitellään yksittäisen olemassa olevan tietueen sisältämää tietoa, oli kyse sitten muokkaamisesta tai lukemisesta. Taulukon indekseistä pysytään kärryillä asettamalla ne dialogi-ikkunan listanäkymän riveille tiedoiksi LB_SETITEMDATA-viestin avulla. Dialogi-ikkunasta ja sen toiminnasta kerrotaan lisää tuonnempana dialogi-ikkunoiden toiminnallisen kuvauksen yhteydessä. Tiedoille voidaan siis suorittaa operaatiot, joilla haetaan yksittäisen tietueen nimi, versionumero ja tiedostonimi, sekä muokata samoja tietoja. Funktioilla, jotka eivät tarvitse parametrinaan indeksiä, muokataan ja haetaan tietoa yleisellä tasolla. Tällaisia operaatioita ovat talletettujen tietueiden lukumäärän pyytäminen, uuden tietueen lisääminen ja aiemmin mainitun lukon omistajan selvittäminen.

Käyttäjäkohtaisten COM/IP-asetusten hallintaan on myös oma luokkansa, userSettings. Asetukset ovat aina u_info.txt-nimisessä tiedostossa asennushakemistossa, josta niitä myös yritetään lukea. Tiedoston ensimmäisten rivien on oltava oikeassa järjestyksessä, sillä makrot lukevat vain kaksi ensimmäistä riviä ja olettavat saavansa oikeaa tietoa. Kahdelle ensimmäiselle riville kirjoitetaan siis aina valitut asetukset. Loput tallennetut asetukset kirjoitetaan niiden jälkeen puolipisteillä eroteltuna ja merkitään viimeksi valittu profiili, jotta se osataan valita, kun sovellus käynnistetään.

Luokka sisältää funktiot getProfileName, getCOMPort ja getIp, jotka palauttavat merkkijonona profiilin nimen, konsoliportin numeron ja ip-osoitteen. Ne ottavat parametrinaan vastaan indeksin, jolla oikean profiilin tiedot osataan lukea. Tässäkin luokassa profiilit säilötään taulukossa tietueina. Indeksiä tarvitsevat myös funktiot setDefault ja getDefault. Niillä selvitetään, mikä profiili on ollut viimeksi valittuna. Se profiili, joka palauttaa getDefault-arvon true asetetaan käytettäväksi profiiliksi. Jos sellaista profiilia ei löydy, ei valita mitään profiilia. Aina kun käyttäjä vaihtaa profiilia, kutsutaan setDefault-funktiota valitun profiilin indeksillä. Näiden lisäksi on vielä delProfile, jota käytetään profiilien poistamiseen indeksin perusteella. COM/IP-profiileja ei siis voi lainkaan muokata, ainoastaan lisätä ja poistaa. Tämä siitä syystä, että uuden profiilin lisääminen on hyvin yksinkertaista ja dialogi-ikkunasta olisi täytynyt tehdä monimutkaisempi, jotta myös tietojen muokkaaminen olisi mahdollista. Ohjelmistoversioiden hallintaan käytettävä dialogi-ikkuna muistuttaakin COM/IP-profiilien hallintadialogia hyvin paljon sillä erotuksella, että ohjelmistoversioita on mahdollista myös muokata.

Kaikki konfiguraatiotiedostoja lukevat luokat antavat virheilmoituksen, mikäli niiden tarvitsemaa konfiguraatiotiedostoa ei löydy tai siinä on virheitä. Btnconfig-luokan virheilmoitukset ovat kaikkein kattavimpia, sillä se on vastuussa useimpien konfiguraatiotiedostojen lukemisesta. Esimerkiksi makrot sisältävän konfiguraatiotiedoston virheet on hyvä esittää niin, että käyttäjä tietää, mistä päin tiedostoa virhettä tulisi etsiä. Ne annetaankin niin, että kerrotaan, mikä oli virheen syy ja millä rivillä virhe ilmaantui.

4.2.4 Toiminta ja pääluokka

Sovelluksen toiminta on varsin suoraviivaista. Konfiguraatiotiedoston lukemisen jälkeen ohjelmalla on muistissaan taulukoita, jotka pitävät sisällään painikkeiden tiedot tietueissa sekä tietueiden lukumäärän. Painikkeita piirrettäessä kysytään btnconfig-oliolta painikkeiden kokonaismäärä, ja käytetään tätä lukua sellaisen for-silmukan toistoehtona, jossa on toinen silmukka sisällä. Ensimmäisessä silmukassa täytetään pääluokassa taulukot, joihin säilötään HWND-tyyppiset kahvat tuleviin ikkunoihin eli painikkeisiin. Sisemmässä silmukassa jokaisella ulomman silmukan kierroksella kysytään iteraation numeroa indeksinä käyttäen btnconfig-oliolta painikkeen nimi ja sijainti. Nimi näytetään piirrettävän painikkeen tekstinä ja sijainnin perusteella päätetään, mihin kohtaan ruutua painike tulee piirtää. Piirtoalue on jaettu riveihin ja sarakkeisiin, ja btnconfig-olio pitää huolen siitä, että painikkeelle ei ole määritelty sijaintia, joka jää sallitun alueen ulkopuolelle. Samaan kohtaan ei myöskään voi määritellä kahta painiketta, vaan se tuottaa virheilmoituksen ja vain ensimmäinen kyseiselle paikalle määritelty painike piirretään ruudulle. For-silmukan iteraation numeroa käytetään myös painikkeen lähettämän viestin wParam-arvon määrittämiseen niin, että arvoksi tulee makropainikkeiden kohdalla iteraation numero + 1, eli 1, 2, 3 jne. Linkkien kohdalla arvoksi määritetään iteraatio + 55, eli 55, 56, 57 jne. Tämän arvon perusteella ikkunan ikkunaproseduuri tietää minkä tyyppisestä painikkeesta on kyse ja kuinka sen painallukseen on reagoitava. Painikkeet piirretään CreateWindow()-funktiolla, jonka palauttama HWND-kahva sijoitetaan aiemmin täytetyn taulukon alkioon. Kahvoja käytetään myöhemmin kun painikkeet halutaan poistaa kutsumalla DestroyWindow()-funktiota. Painikkeiden piirtämisen jälkeen käynnistetään ajastin, joka päivittää painikkeet automaattisesti 15 minuutin välein, jotta käyttäjän näkymä pysyy ajan tasalla.

Kun painikkeet on piirretty, sovellus on toimintakunnossa. Painikkeen painallus aiheuttaa WM_COMMAND-viestin, johon reagoidaan välittämällä viestin wParam-parametri BtnAction()-funktiolle. Funktio käyttää parametria indeksinä, jolla se kysyy btnconfig-luokan ilmentymältä painikkeen tietoja. Painikkeen indeksi btnconfig-olion taulukossa on makropainikkeiden kohdalla wParam-1, ja linkkien kohdalla wParam-55. Näin taulukosta osataan hakea aina oikeaa painiketta vastaavat tiedot.

Makron suorittamiseen tarvittavat painikkeen tiedot ovat toiminnallisuus sekä työkansio. Makro suoritetaan ShellExecute()-funktiolla, jolle annetaan parametreina polku ttpmacro.exe-tiedostoon, käynnistysparametriksi haettu toiminnallisuus, ja työkansioksi haettu työkansio. Näin saadaan käynnistettyä ttpmacro.exe parametrinaan oikea makro niin, että makron työhakemisto on sama kuin se, jossa kyseinen makro sijaitsee.

```
//nappien toiminnallisuus
void btnAction(int wmId){
    //nappuloiden indeksit 0, 1, 2 jne
    wstring bAction = btnObject.getAction((wmId - 1), false);
    wstring bDirectory = btnObject.getWorkingDir((wmId - 1), false);
    ShellExecute(NULL, L"open", teraterm.c_str(), bAction.c_str(), bDirectory.c_str(), SW_SHOWNORMAL);
}
//linkkien toiminnallisuus
void linkAction(int wmId){
    wstring lAction = btnObject.getAction((wmId - 55), true);
    LPCWSTR linkAction = (LPCWSTR)lAction.c_str();
    ShellExecute(NULL, L"open", linkAction, NULL, NULL, SW_SHOWNORMAL);
}
```

Kuva 6. Painikkeen toiminnallisuuden suoritus

Kuva 6 esittää sovelluksen funktioita, joilla suoritetaan makrot ja avataan linkit. Linkkien tapauksessa ShellExecute-funktiolle ei tarvitse antaa työhakemistoa, mutta makropainikkeiden kohdalla se on välttämätöntä. Ilman työhakemistoa makro käynnistyy luullen, että niitä suoritetaan esimerkiksi C:\-aseman juuresta, eivätkä täten osaa avata tiedostoja, joihin viitataan suhteellisella tiedostopolulla. Työhakemistoa ei tarvitse konfiguroida makropainikkeiden konfiguraatiotiedostoon erikseen, vaan ne luetaan jokaisen makron polusta poistamalla siitä tiedoston nimi. Työhakemistoksi tulee siis aina sama hakemisto kuin se, jossa suoritettava makro sijaitsee. Työhakemisto irrotetaan tiedostopolusta btnconfig-luokan funktiolla parseWorkingDir, joka pilkkoo saamansa merkkijonon kenoviivojen kohdalta ja palauttaa sen ilman viimeistä palaa. Esimerkiksi ”C:\Program Files\Kansio\makro.ttl” palautuu muodossa ”C\:Program Files\Kansio\”.

Painikkeiden päivittämisen yhteydessä kaikki painikkeet poistetaan ja luetaan uudestaan tiedostosta. Painikkeet poistetaan käymällä pääluokan painikkeista muodostama

taulu läpi ja kutsumalla DestroyWindow()-funktiota jokaiselle luodulle painikkeelle. Tämän jälkeen taulukko tyhjennetään ja kutsutaan btncobject-luokan resetButtons()-funktiota. ResetButtons nollaa painikkeiden lukumäärän, tyhjentää taulukot, joihin painikkeiden tiedot on luettu, ja lukee tiedot uudestaan tiedostosta muistiin kuten aivan alussakin. Tämän jälkeen pääluokassa kysytään jälleen painikkeiden lukumäärää ja iteroidaan uudet painikkeet eli ikkunat taulukkoon, jonka jälkeen piirretään uudet painikkeet näkyviin.

Dialogi-ikkunat, joista kerrotaan lisää seuraavassa osiossa, saadaan näkyviin pääikkunan valikkopalkista. Valikon kohdassa Settings on listattu kaikki tallennetut COM/IP-profiilit sekä painikkeet, joilla saadaan avattua ohjelmistoversioiden sekä COM/IP-profiilien hallintadialogit. Valikkopalkkia päivitetään aina kun COM/IP-profiili lisätään tai valitaan uusi profiili. Myös sovelluksen pääikkunan otsikkopalkin tekstiä muutetaan profiilin valinnan yhteydessä, jotta siihen saadaan näkyviin kulloinkin valittu profiili.

4.2.5 Dialogi-ikkunat

Kuten aiemmin mainittiin, esaM-sovellus käyttää muutamia dialogi-ikkunoita käyttäjän syötteen vastaanottamiseen ja talletettujen tietojen selailuun. Dialogi-ikkunoilla on kaikilla omat ikkunaproseduurinsa, jotka vastaavat kunkin dialogin viestien käsittelystä. Pääikkunasta poiketen dialogi-ikkunoiden ulkoasua ei ole kirjoitettu ohjelmakoodiin käsin, vaan ne on luotu käyttäen Visual Studion resurssieditoria. Dialogi-ikkunoita on sovelluksessa neljä. Kaksi niistä on ohjelmistoversioiden hallintaan käytettäviä ikkunoita, joista toisella selataan ja muokataan talletettuja tietoja, ja toisella saadaan lisättyä uusia. Yksi dialogi-ikkuna on COM/IP-profiilien hallintaa varten ja viimeinen Visual Studion generoima About-ikkuna, jolla silläkin on oma ikkunaproseduurinsa. Kuvissa 4 ja 5 näkyy dialogi-ikkunat joilla hallitaan COM/IP-profiileja sekä ohjelmistoversioita.

COM/IP-profiilien ja ohjelmistoversioiden hallintadialogit näyttävät tietoa listoissa, joihin ne hakevat oikean luokan ilmentymältä tiedot heti ikkunan luonnin yhteydessä[15]. Tämä tapahtuu dialogien ikkunaproseduureissa silloin, kun vastaanotetaan viesti WM_INITTIALOG. Tiedot saadaan näytettyä listoissa lähettämällä listalle viesti LB_ADDSTRING. Listat järjestellään automaattisesti, joten haettujen tietojen indeksit on otettava talteen ja sidottava listan riveihin viestillä LB_SETITEMDATA[16]. Listalla näkyvä järjestys ei siis ole sama kuin se, jossa tiedot on alun perin luettu tiedostosta.

Dialogien ikkunaproseduurien tehtävänä onkin hallita kaikkea, mitä dialogi-ikkunoissa tapahtuu. Tällaisia tapahtumia ovat erinäisten painikkeiden painaminen sekä listalta valinta. Tapahtumiin tulee reagoida tilanteen vaatimalla tavalla. Esimerkiksi ohjelmistoversioiden hallintadialogin ikkunaproseduurin tulee reagoida ”Cancel”- ja ”Ok”-painikkeisiin seuraavalla tavalla: Molemmat painikkeet aiheuttavat sen, että kutsutaan swVersionConfig-luokan funktiota writeConfig (bool lockfile, bool rollback) sellaisilla parametreilla, että lopputulos on toivottu. ”Cancel”-painikkeen tapauksessa toiseksi parametriksi asetetaan true, jolloin swVersionConfig-luokan ilmentymä kirjoittaa tiedostoon täsmälleen samat tiedot, jotka se sieltä aiemmin luki. ”Ok”-painikkeen tapauksessa annetaan parametri true, jolloin tiedostoon kirjoitetaan dialogi-ikkunaan muokatut tai lisätyt tiedot. Ohjelmistoversioiden hallintadialogi myös tarkistaa onko ohjelmistoversiot sisältävä tiedosto lukittu, ja jos on niin se pyytää swVersionConfig-luokan ilmentymältä tiedon lukon haltijasta. COM/IP-profiilien hallintadialogi toimii hyvinkin pitkälti samaan tapaan kuin ohjelmistoversioiden. Se on toiminnaltaan hieman yksinkertaisempi, sillä sen avulla ei voi muokata olemassa olevaa tietoa. Sillä voi ainoastaan lisätä ja poistaa tietoa, joka taas tekee ikkunassa näkyvän listan hallinnasta helpompaa. Ohjelmistoversioiden hallintadialogin muokkausominaisuuden vuoksi tarvitaan yksi kokonaan erillinen dialogi uuden ohjelmistoversiotiedon lisäämiseen. Tämän dialogin saa näkyville ainoastaan ohjelmistoversioiden muokkausdialogin kautta, ja sen ainoa tarkoitus on pyytää käyttäjältä uuden ohjelmistoversion laitetunnus, ohjelmiston versionumero sekä ohjelmatiedoston nimi. Kuvassa 5 nähdään myös tämä dialogi-ikkuna. Samasta kuvasta voidaan varsinkin kuvaan 4 vertaamalla huomata, että dialogi-ikkunasta tulisi huomattavasti vaikeaselkoisempi, jos samalla näkymällä voisi lisätä ja muokata tietoja. Silloin näkyvillä pitäisi olla lisäämiseen käytettävät tyhjät kentät, joita kuitenkin käytetään suhteellisen harvoin, sillä esikonfiguroinnissa käytettäviä ohjelmistoversioita ei tule kovin usein uusia.

4.3 Käyttöönotto ja testaus

Sovelluksen käytännön testaaminen osoittautui kehitysvaiheessa haastavaksi, sillä kehitysympäristö oli hyvin erilainen kuin lopullinen käyttöympäristö. Kehitysympäristössä käytössä oli Windows 10, kun taas käyttöympäristössä käytössä oli Windows 7. Käyttöympäristön tietokoneet olivat myös hieman hitaampia kuin ohjelmoinnissa käytettä tietokone. Myöskään verkkolevyä ei ollut kehitysympäristössä käytössä, joten sen mahdolliseen temppuiluun ei voinut varautua etukäteen. Edellä mainituista syistä ja tiukasta aikataulusta johtuen käytännön testaaminen jäi suurelta osin loppukäyttäjien

harteille. Kun sovelluksesta valmistui kehitysversio, se siirrettiin käyttäjien saataville ja käytettäväksi. Tällä tavalla ilmeni ohjelmointivirheitä ja muita virhetilanteita, joita ei kehitysvaiheessa tullut edes ajatelleeksi. Ilmenneet viat kirjattiin ylös ja niiden syyt selvitetiin. Yksi esimerkki tällaisesta oli Windowsin fontit, ja niiden toiminta, johon en ollut aiemmin perehtynyt juurikaan. Ohjelmointiympäristössä kokeiltuna ongelmaa ei ilmennyt, mutta käyttäjille annettaessa sovelluksen fontit toimivatkin täysin arvaamattomasti. Syyksi osoittautui se, että käyttäjillä oli erilaisia DPI-asetuksia työasemillaan. Tämä aiheutti sen, että tekstit eivät mahtuneet painikkeisiin niin kuin oli suunniteltu. Ongelma korjaantui skaalaamalla fonttikoko niin, että se on aina samankokoinen pikseleissä riippumatta siitä, mikä on käyttöjärjestelmän DPI-asetus. Toinen vaihtoehto olisi ollut skaalata koko käyttöliittymä järjestelmän asetusten mukaan, mutta kun koko muu käyttöliittymä perustui pikseleissä mitattuihin kokoihin, päädyin helpompaan ja nopeampaan ratkaisuun, eli fonttien skaalaamiseen. Näin käyttöliittymä toimii ja näyttää samalta kaikilla asetuksilla. Toinen asia, jota en ottanut huomioon oli se, että Windowsin eri versioissa ikkunoiden reunat ovat eri paksuisia. Windows 10 -käyttöjärjestelmällä toimivaksi testattu sovellus ei ollutkaan 7-versiolla oikean näköinen, sillä Windows laskee ikkunan kokoon myös reunat, ja tämä sotki sovelluksen pikseleihin perustuvan ulkoasun täysin. Funktioilla AdjustWindowRectEx ja SetWindowPos saadaan ensin laskettua tarvittava ikkunan koko, kun tiedetään, minkä kokoinen ikkunan piirtoalueesta halutaan, ja sen jälkeen muutettua ikkunan kokoa vastaamaan toivottua piirtoaluetta.

Myös näkymän automaattisesta päivittämisestä aiheutui ongelmia. Ominaisuus lisättiin siksi, että järjestelmään lisätyt makrot eivät päivittyneet käyttäjille, elleivät he painaneet päivityspainiketta. Ominaisuus kuitenkin aiheutti ongelmia siinä vaiheessa, jos yhteys verkkolevyyn oli katkennut. Tällaisessa tilanteessa konfiguraatiotiedoston lukeminen ei onnistu, vaan sovellus antaa virheilmoituksen. Kun tätä jatkuu esimerkiksi viikonlopun yli 15 minuutin välein, on virheilmoituksia kertynyt ruudulle aika liuta. Ainoa tapa toipua tilanteesta onkin lopettaa sovellus tehtävienhallinnan kautta. Tähän kehitettiin ratkaisuksi se, että jokaisessa virheilmoituksia tuottavassa luokassa on myös funktio, joka kertoo, onko virheilmoitus jo annettu. Näin uutta virheilmoitusta ei näytetä, jos taustalla on jo vanha virheilmoitus. Tässäkin ratkaisussa on kuitenkin omat ongelmansa sen suhteen, miten tämä virheilmoituksen kuittaaminen rekisteröidään sovelluksessa niin, että sen sallitaan taas näyttää virheilmoituksia. Raportin kirjoitushetkellä ongelmaa ei ole korjattu käytössä olevaan versioon vaan se sisältyy sovelluksen jatkokehitykseen.

Toiminnan siirtyessä yritykseltä toiselle, myös käytössä ollut verkkolevy muuttui. Uusi verkkolevy saatiin käyttöön jonkin aikaa ennen varsinaista uusiin toimitiloihin siirtymis-

tä, joten sovellusta päästiin testaamaan ja laittamaan käyttökuntoon hyvissä ajoin ennen toiminnan käynnistymistä. Konfiguraatiotiedostoihin perustuva toteutus osoittautui lopulta hyväksi ratkaisuksi. Sovelluksen kaikkine oheistiedostoineen pysyi siirtämään paikasta toiseen varsin pienillä muutoksilla, eikä lähdekoodiin tarvinnut kajota lainkaan, mikä olikin kehityksessä pääpainona.

Muutoksen yhteydessä kävi kuitenkin ilmi, että makroissa oli varsin runsaasti tiedostopolkuja, jotka täytyi muuttaa, jotta ne toimivat siirron jälkeen oikein. Osan tiedostopoluista sai kirjoitettua suhteelliseen muotoon, eli niin että tiedostopolussa on vain polku nykyisestä sijainnista lähtien. Esimerkiksi polku C:\Kansio\Makrokansio\Alempikansio\tiedosto.txt voitiin lyhentää muotoon ”./Alempikansio/tiedosto.txt”. Muutosten jälkeen myös makrot saatiin sellaisiksi, että niitä voi siirtää paikasta toiseen tekemättä juurikaan muutoksia, ja ne toimivat aivan kuten ennenkin.

Samassa yhteydessä siirryttiin Tera Termin uudempaan versioon, jotta jatkossa voidaan hyödyntää siihen lisättyjä komentoja, jotka esimerkiksi merkkijonojen käsittelyssä ovat huomattavasti edellä sitä versiota, joka aiemmin oli kaikilla asennettuna. Päivityksen yhteydessä makroihin oli tehtävä vielä lisää muutoksia, sillä uudemmissa Tera Termin versiossa täytyy merkkijonot syöttää komentoon ”strspecial”, jotta niihin voi sisällyttää ääkkösiä tai erikoismerkkejä kuten rivinvaihtoja. ”strspecial” toimii myös aiemmin käytössä olleessa versiossa, joten makrojen muuttaminen ei vaikuta niihinkään käyttäjiin, jotka syystä tai toisesta päättivät olla päivittämättä Tera Termiä uudempaan versioon [17].

5 Yhteenveto

Toiminta uusissa tiloissa käynnistyi neljäs päivä huhtikuuta, ja esaM-sovellus saatiin käyttöön ilman suurempia ongelmia. Tuolloin asennettavissa ollut sovelluksen versio oli niin sanotusti viimeistä edellinen, eli siihen ei ollut tehty viimeisiä viilauksia. Uusia ominaisuuksia ohjelmaan ei viimeiseen versioon tullut lisää, eikä käyttäjä oikeastaan edes voi erottaa uudempaa ja vanhempaa versiota toisistaan. Sovelluksen koodia on lähinnä siistitty ja yksinkertaistettu. Joitakin silmukoita on paranneltu, muutettu sovellus käyttämään pelkkiä unicode-merkkijonoja ja etsitty kaikki turhat muunnokset ja muuttujat. Koodiin on lisätty myös kommentteja ja vaihdettu joidenkin muuttujien nimiä niin, että koodi olisi ulkopuoliselle lukijalle selkeämpää lukea.

Työn tavoitteena oli kehittää datalaitteiden esikonfigurointia edelleen niin, että se olisi mahdollisimman helppoa ja vaivatonta. Tavoite oli varsin kunnianhimoinen, eikä sen voida vielä sanoa täyttyneen. Vaikka esikonfigurointi saattaa vaikuttaa suhteellisen mutkattomalta prosessilta, on siinä kuitenkin hyvin paljon ennustamattomissa olevia muuttujia. Konfiguroitavat laitteet eivät aina käyttäydy niin kuin niiden pitäisi, jolloin käyttäjältä vaaditaan asiantuntemusta. Niin kauan, kun mikään ei mene pieleen, voidaan sanoa, että makrojen ja kehitetyn sovelluksen avulla laitteita voi esikonfiguroida tietämättä niiden toiminnasta mitään. Riittää, että tunnistaa konfiguroitavan laitteen ja osaa yhdistää sen oikeaan nappulaan sovelluksen näkymässä.

Käytettävien makrojen osalta kehitystyötä ei saatu vietyä vielä loppuun asti, sillä makrojen testaaminen vie aikaa. Koska kyseessä on asiakasyritys ja sen laitteet, ei myöskään käy päinsä testailla tuotantoon menevillä laitteella uusia makroja, vaan testausta varten olisi varattava erikseen aikaa, jotta varsinainen esikonfigurointityö ei jää tekemättä. Niiltä osin kun makroja kuitenkin ehdittiin kehittää, voidaan sanoa, että kehitysprojekti oli onnistunut. Makrojen kehittämisessä on otettava huomioon myös se, että niitä käyttää moni muukin, eikä niistä tule kirjoittaa sellaisia, ettei kuka tahansa muu asiaa yhtään tunteva osaisi sanoa, mitä ne tekevät. Jos makrosta kirjoittaa mielestään hyvän ja toimivan, muttei dokumentoi sitä lainkaan, voi jollakin mennä sormi suuhun vaikkapa vuosia myöhemmin, kun on tarve tehdä makroon joitain muutoksia. Makrojen osalta suurin ongelma oli kuitenkin se, että niiden kehityksen kuvaaminen tähän raporttiin osoittautui erittäin vaikeaksi. Asiakasyritys ei halunnut, että raporttiin päätyy mitään laitetietoja tai konfiguraatioita, joten makrojen kuvaaminen käytännön esimerkein oli mahdotonta. Makrojen kehitys oli kuitenkin työmäärältään suurempi osa kehitystyötä kuin niiden käynnistämiseen tarvittava sovellus, joten raportti saattaa antaa hieman väärän kuvan kehitystyön luonteesta. Tästä syystä raportissa päädyttiin kuvaamaan kehitetyn sovelluksen rakennetta tarkemmin ja makrojen toimintaa kuvattiin vain yleisellä tasolla.

Kehitettävän sovelluksen osalta projekti oli onnistunut. Sovelluksen käyttö on suoraviivaista. Uskaltaisin väittää, että sitä osaa käyttää lähes kuka tahansa kaduntallaaja. Suurin osaamista vaativa asia sovelluksessa onkin sen asennus. Konfiguraatiotiedostot pitää osata kirjoittaa käytettävän ympäristön mukaisesti käsin. Tätä voisi jatkossa kehittää niin, että sovellus kysyy tarvittavat tiedostopolut käyttäjältä ensimmäisen käynnistyksen yhteydessä ja toimii jatkossa niiden perusteella. Kehittäjälleen sovelluksen teko opetti paljon uutta sekä Windows-ohjelmoinnista että C++ kielestä. Yksi syy ohjelmointikielen valintaan olikin halu oppia ymmärtämään ja kirjoittamaan Windows-sovelluksia.

Myönnettäköön, etten etukäteen ollut tajunnut, kuinka vähän aiheesta oikeastaan tiedän. En voi vieläkään väittää olevani alan taitaja, mutta kehitysprojektin aikana kiinnostus aihetta kohtaan kasvoi entisestään, eikä minun tarvitse seuraavankaan ohjelmointiprojektin kohdalla kauaa miettiä, minkä kielen ja ympäristön valitsen. Tämän projektin jatkokehityksen aikana uskon oppivani Windows-ohjelmoinnista vielä paljon lisää, ja voikin olla että tämä jo useaan kertaan uudelleen kirjoitettu koodi menee vielä kerran jos toisenkin uusiksi. Projektia voikin kutsua ikuisuusprojektiksi, sillä en usko sen olevan koskaan siinä tilassa, että sen voisi sanoa olevan valmis. Aina löytyy jotain kehitettävää: niin kehitetyn sovelluksen kuin esikonfiguroinnissa käytettävien makrojenkin osalta.

Kehitysprojekti onnistui tiukasta aikataulusta huolimatta varsin mutkattomasti. Sovellus kaikkine oikeuksineen luovutettiin Posti Oy:lle 12.4.2016. Lisäominaisuuksia sovellukseen kehitetään, kun ideoita tai tarpeita ilmenee ja havaittuja ongelmia korjataan sitä mukaa, kun niitä löytyy.

Lähteet

1 TTL Command Reference. Verkkodokumentti. https://ttssh2.osdn.jp/manual/en/macro/command/wait.html. Luettu 12.3.2016.

2 TTL Command Reference. Verkkodokumentti. https://ttssh2.osdn.jp/manual/en/macro/command/. Luettu 12.3.2016.

3 TTL Command Reference. Verkkodokumentti. https://ttssh2.osdn.jp/manual/en/macro/command/connect.html. Luettu 12.3.2016.

4 TTL Command Reference. Verkkodokumentti. https://ttssh2.osdn.jp/manual/en/macro/command/strsplit.html. Luettu 14.3.2016.

5 Petzold, Charles 1998. Programming Windows Fifth Edition. Microsoft Press. Luku 3, sivu 34.

6 About Messages and Message Queues. Windows Dev Center. Verkkodokumentti. https://msdn.microsoft.com/en-us/library/windows/desktop/ms644927%28v=vs.85%29.aspx#windows_messages MSDN. Luettu 10.4.2016.

7 Windows Data Types. Windows Dev Center. Verkkodokumentti. https://msdn.microsoft.com/en-us/library/windows/desktop/aa383751%28v=vs.85%29.aspx. Luettu 12.4.2016.

8 WM_COMMAND message. Windows Dev Center. Verkkodokumentti. https://msdn.microsoft.com/en-us/library/windows/desktop/ms647591%28v=vs.85%29.aspx. Luettu 12.4.2016.

9 WM_CREATE message. Windows Dev Center. Verkkodokumentti. https://msdn.microsoft.com/en-us/library/windows/desktop/ms632619%28v=vs.85%29.aspx. Luettu 12.4.2016.

10 Using Timers. Windows Dev Center. Verkkodokumentti. https://msdn.microsoft.com/en-us/library/windows/desktop/ms644901%28v=vs.85%29.aspx. Luettu 26.4.2016.

11 Using Window Procedures. Windows Dev Center. Verkkodokumentti. https://msdn.microsoft.com/en-us/library/windows/desktop/ms633570%28v=vs.85%29.aspx#designing_proc. Luettu 15.4.2016.

12 IP Address Control. Windows Dev Center. Verkkodokumentti. https://msdn.microsoft.com/en-us/library/windows/desktop/bb761374%28v=vs.85%29.aspx. Luettu 20.4.2016.

13 Getenv_s, _wgetenv_s. Microsoft Developer Network. Verkkodokumentti. https://msdn.microsoft.com/en-us/library/tb2sfw2z.aspx. Luettu 14.4.2016.

14 Wstring. Microsoft Developer Network. Verkkodokumentti. https://msdn.microsoft.com/en-us/library/wt3s3k55.aspx. Luettu 21.4.2016.

15 List Box. Windows Dev Center. Verkkodokumentti. https://msdn.microsoft.com/en-us/library/windows/desktop/bb775146%28v=vs.85%29.aspx. Luettu 15.4.2016.

16 List Box Messages. Windows Dev Center. Verkkodokumentti. https://msdn.microsoft.com/en-us/library/windows/desktop/ff485967%28v=vs.85%29.aspx. Luettu 15.4.2016.

17 TTL Command Reference. Verkkodokumentti. https://ttssh2.osdn.jp/manual/en/macro/command/strspecial.html. Luettu 23.4.2016